LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-PROC-208680

Multi-Resolution Representation of Topology

K. Cole-McLaughlin, V. Pascucci

December 17, 2004

IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP 2004)
Marbella, Spain
July 6, 2004 through July 8, 2004

MULTI-RESOLUTION REPRESENTATION OF TOPOLOGY

Kree Cole-McLaughlin Valerio Pascucci*
Ceneter for Applied Scientific Computing, Lawrence Livermore National Laboratory

Abstract

The Contour Tree of a scalar field is the graph obtained by contracting all the connected components of the level sets of the field into points. This is a powerful abstraction for representing the structure of the field with explicit description of the topological changes of its level sets. It has proven effective as a data-structure for fast extraction of isosurfaces and its application has been advocated as a user interface component guiding interactive data exploration sessions. We propose a new metaphor for visualizing the Contour Tree borrowed from the classical design of a mechanical orrery – see Figure 1(a) – reproducing a hierarchy of orbits of the planets around the sun or moons around a planet. In the toporrery – see Figure 1(b) – the hierarchy of stars, planets and moons is replaced with a hierarchy of maxima, minima and saddles that can be interactively filtered, both uniformly and adaptively, by importance with respect to a given metric.

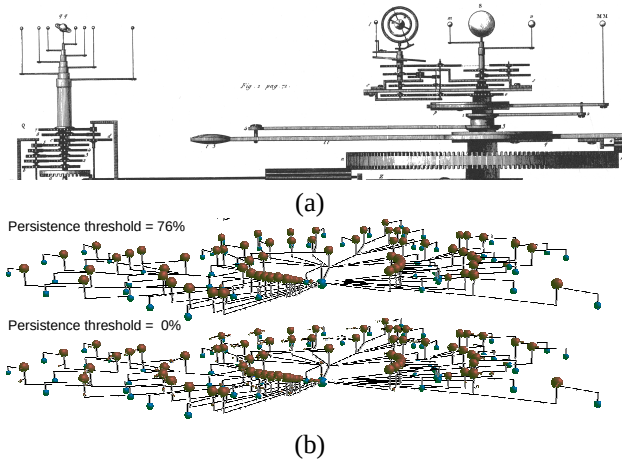


Figure 1. (a) Orrery reproducing the hierarchical relationship between the orbits of the sun, the planets and their moons. Original design (1812) by A. Janvier reprinted recently by E. Tufte [20]. (b) Toporrery representing the hierarchical relationship between the critical points in a scalar field both at full resolution and reduced at 76% of persistence.

1 Introduction

A Morse function over a domain $\mathcal{D}$, is a smooth mapping, $f : \mathcal{D} \rightarrow \mathbb{R}$, such that all its critical points (maxima, minima and saddles) are distinct. Complex natural phenomena, both sampled and simulated, are often modeled as Morse functions[1]. MRI scans generate Morse functions that are used in medical imaging to reconstruct human tissues. Electron density distributions computed by high-resolution molecular simulations are Morse functions whose topology express bonds among the atoms in molecular structures. The structure of geometric models used in computer graphics and CAD applications can be effectively represented in terms of the topology of a Morse function [9].

The Reeb graph [13] is a simple structure that summarizes the topology of a Morse function. For functions with simply connected domains this graph is also simply connected and is called the Contour Tree. The Reeb graph has been used to analyze the evolution of teeth contact interfaces in the chewing process [14], and to compute indices of topological similarity for databases of geometric models [9]. Topological information has been used to guide the construction of transfer function for volume rendering of scientific data [17, 22]. A more extensive discussion of the use of the Reeb graph and its variations in geometric modeling and visualization can be found in [8].

The first algorithm for constructing Reeb graphs of Morse functions with two-dimensional domains is due to [15]. Given a triangulated surface, this scheme takes as input the set of all distinct level lines and therefore has worst case time complexity $O(n^2)$, where n is the number of vertices in the triangulation. An $O(n \log n)$ algorithm for computing contour trees in any dimension was introduced in [4]. This scheme has been extended in three dimensions to include the genus of all isosurfaces [12]. The first multi-resolution representation of the Reeb graph was introduced in [9]. Their method hierarchically samples the range space of f while concurrently refining the Reeb graph. They obtain a multi-resolution model that is suitable for fast comparison of graphs. However, this hierarchy does not represent the topology of f at multiple levels of detail. A formal framework for ranking topological features by persistence has been introduced in [7] and applied to two-dimensional Morse functions in [1]. Topological simplification is used in [16] to design transfer function that highlight only the major features in the data. Topological simplification is also widely used in vector field visualization to highlight the most important structures present in the data [19, 18].

Integration of the Contour Tree in user interfaces to help selecting isosurfaces has been first suggested [21] but was only fully developed six years later in [2]. The lat-

[1]Technically the definition of Morse function is often weakened to allow multiple critical points or other degeneracies present in real data.

*This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

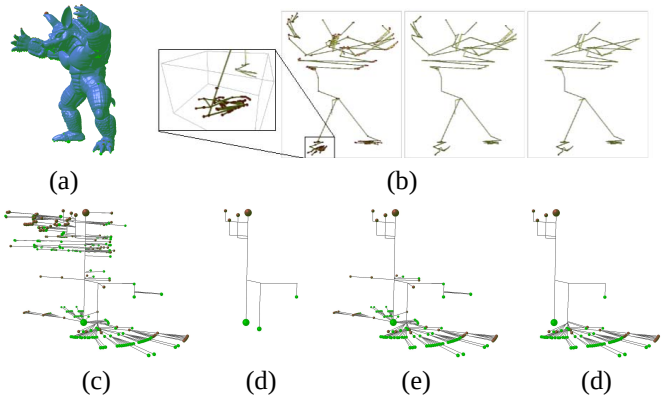


Figure 2. (a) A polygonal armadillo model. The Morse function f is the height in the vertical direction. The critical points are marked with small spheres. (b) The contour tree of f presented with the critical points in their original position. Several version of the tree with adaptive (one foot) or uniform refinement. (c) Full resolution toporrery of f. (d) Simplification down to 58% of persistence. Adaptive refinement with full resolution for lower half of the body (e) or only the left foot (f).

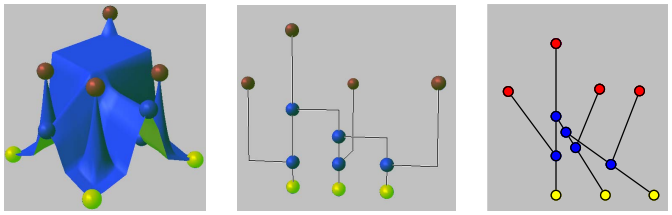


Figure 3. (a) A simple terrain model. The Morse function f is vertical elevation. The critical points are highlighted with spheres: red for maxima, yellow for minima and blue for saddles. (b) Toporrery of f. (c) Planar layout of the same tree.

ter work is particularly interesting for the use of a new concept of "Path Seeds" that links explicitly the arcs in the Contour Tree to distinct connected components (contours) of the level sets. This introduces the powerful new paradigm of selecting contours instead of entire isosurfaces. Our scheme introduces a user interface based on a multi-resolution representation of the Contour Tree (not just a simplification) and a metaphor for presenting the the tree in a 3D abstract embedding similar to a mechanical orrery. In particular, we adapt a simple radial graph drawing algorithm [6] and combine it with an embedding typically used to large tree hierarchies [10].

Our results are summarized in Figures 1 and 2 with presentations of the topology of two scalar fields defined on 3D domains (electron density distribution of water at high pressure) and 2D domains (armadillo).

2 Multi-resolution Contour Trees

Contour Tree of a Morse Function. Let $\mathcal{D}$ be a triangulated domain and $f : \mathcal{D} \to \mathbb{R}$ be a function obtained by linear interpolation of the value of f at the vertices of $\mathcal{D}$. Morse theory provides a formal framework for understanding the topology of $\mathcal{D}$ by analyzing the function f. The fundamental tool in Morse theory is the characterization of each point of $\mathcal{D}$ as being either regular or critical.

We assume that $\mathcal{D}$ is a simplicial complex. Therefore, every k-cell c of $\mathcal{D}$ is the convex hull of $k+1$ vertices of $\mathcal{D}$. Moreover, a cell c' is a called *face* of c if its vertices are a subset of those of c. If $c \in \mathcal{D}$ then all its faces must be in $\mathcal{D}$. For a vertex $v \in \mathcal{D}$, its *link* Lk_v is the set of cells that do not contain v but that are faces of some cell containing v. Furthermore, the *lower link* of v, Lk_v^-, is the set of all cells in Lk_v that have only vertices with function value smaller than $f(v)$. The *upper link* Lk_v^+ is the set of cells in Lk_v that have only vertices with function value greater than $f(v)$.

Definition 1 *Let $\mathcal{D}$ be a triangulated manifold with boundary and $f : \mathcal{D} \to \mathbb{R}$ be a piecewise-linear function. A vertex $v \in \mathcal{D}$ is called* regular *if both Lk_v^- and Lk_v^+ have exactly one connected component. Otherwise v is called a* critical point *and $f(v)$ is called a* critical value.

We can now define a Morse function. Since the definition only refers to critical points it applies equally well in the smooth and discrete settings.

Definition 2 *f is a* Morse function *iff all its critical values are distinct.*

On piecewise-linear functions this condition can be enforced by symbolically perturbing the critical values. If the vertices $v_i, v_j \in \mathcal{D}$ are critical points such that $f(v_i) = f(v_j)$, then we define $f(v_i) < f(v_j)$ if and only if $i < j$. In practice we apply the symbolic perturbations to the function value at all the vertices $v \in \mathcal{D}$. This allows us to sort the vertices by their function value and to simply define $f(v_i) \equiv i$.

Definition 3 *A* level set *of f is the pre-image of a real value ω, $L_f(\omega) = f^{-1}(\omega)$. Given a level set, $L_f(\omega)$, we call a connected component of $L_f(\omega)$ a* contour.

Morse theory describes how the the topology of $L_f(\omega)$ changes as the field value, ω, changes. One of the main results states that if a and b are such that the range $[a, b]$ contains no critical values, then $L_f(\omega)$ is homeomorphic to $L_f(\nu)$ for all $\omega, \nu \in [a, b]$. On the other hand if the range $[a, b]$ contains a single critical value, ω_0, then for $\omega \in [a, \omega_0)$ and $\nu \in (\omega_0, b]$ the difference in the topology of $L_f(\omega)$ and $L_f(\nu)$ can be completely described as follows: (i) If ω_0 is a local minimum, a new contour is created in $L_f(\nu)$ that did not exist in $L_f(\omega)$. (ii) If ω_0 is a local maximum, a contour of $L_f(\omega)$ is destroyed. (iii) If ω_0 is a saddle point, either two contours of $L_f(\omega)$ merge into a single contour of $L_f(\nu)$ or one contour of $L_f(\omega)$ divides into two contours of $L_f(\nu)$. For volumetric or higher dimensional domains, a saddle point can also induce a topological change in a single contour of L_f (no split nor merge).

The Contour Tree encodes the changes in the number of contours of the level set.

Definition 4 *Consider the graph obtained by contracting each contour of every level set of f to a point. For general Morse functions this graph is called the* Reeb graph *and can have any number of cycles, depending on the topology of $\mathcal{D}$ [5]. However, if $\mathcal{D}$ is simply connected the* Reeb graph *is also simply connected and is called* Contour Tree.

From the definition it can be seen that the nodes of the contour tree correspond critical points of f and are therefore associated with the relative critical value. Furthermore, nodes that correspond to extrema are leaf nodes, and

nodes that correspond to saddle points must have degree three (or higher in degenerate cases). Figure 3(a) shows a simple terrain as an example of Morse function, where the elevation of each point is the value of f. Figure 3(b) show the corresponding Contour tree. Figure 3(c) shows the planar layout proposed in [21] where the y coordinate of each node is constrained to be equal to to the corresponding critical value of f. Note that with this constraint the graph cannot be drawn in the plane without self-intersections.

Hierarchical Graph Representation We define a multi-resolution representation of the contour tree that allows linear time access to simplified representations of the topology. Typically finite graphs are represented as a list of nodes and a list of arcs, where each arc is defined as a node pair. In this section we discuss an alternative representation called a branch decomposition. A *branch* is a monotone path in the graph traversing a sequence of nodes with non-decreasing (or non-increasing) value of f. The first and last nodes in the sequence are called the endpoints of the branch. All other nodes are said to be interior to the branch. Note that a branch can be thought of equally as a sequence of nodes or a sequence of arcs. A set of branches is called a *branch decomposition* of a graph if every arc the graph appears in exactly one branch of the set. The standard representation of a graph satisfies this definition, where every branch is a single arc. We call this the trivial branch decomposition.

Definition 5 *A branch decomposition of a tree is a* hierarchical tree *if: (i) there is exactly one branch connecting two leaves (called root branch), (ii) every other branch connects one leaf to a node that is interior to another branch.*

We wish to construct a branch decomposition representing the contour tree of a scalar field $f : \mathcal{D} \rightarrow \mathbb{R}$, such that the endpoints of each branch (except the root) represent an extremum paired with a saddle point of the scalar field. See Figure 4. The tree can be simplified by removing a branch that does not disconnect the tree. This corresponds to the cancellation of two critical points in the scalar filed. This simplification process defines a hierarchy of cancellations where a branch $B1$ is said to be the parent of branch $B3$ if one endpoint of $B3$ is interior to $B1$. The root branch has no parent and cannot be simplified. Removal of a parent before one of its children disconnects the tree. In the next section we will discuss the construction of a branch decomposition based on the persistence of critical point pairs.

Once the decomposition is constructed and the parent-child relations are defined, we can build any approximation of the original tree by incrementally connecting child branches to their parent. In particular, we associate values to each branch for several metrics (such as persistence, geometric location or other) and artificially enforce a nesting condition that requires, for all the metrics, the value of the parent to be greater than or equal to the value of its children. Given a tolerance threshold for several metrics at the same time, we start from the root branch and iteratively select children with metrics above the required thresholds.

Tree Layout and Presentation.

We define an embedding of the contour tree, which can be used as a user interface tool. The vertical coordinate-axis is fixed to represent the value of the scalar field. In doing so we loose one degree of freedom, which makes it impossible, in general, to build a planar embedding without self-intersections. Figure 3 is an example of a simple scalar field with a contour tree that cannot be embedded in the plane without self-intersections. Thus, in this section, we describe a three-dimensional embedding of the contour tree that uses the z-coordinate to represent the field value, and such that the projection onto the plane $z = 0$ has no self-intersections. We also provide a progressive construction of this embedding using the multi-resolution representation given above.

Our visualization scheme makes use of an algorithm for the layout of rooted trees [6]. Any such algorithm could be used to produce an embedding. We chose a radial layout algorithm that positions the root node of the tree at the origin and positions its descendants in concentric circles.

The main idea of the layout algorithm is to define a sequence of consecutive disks, $D_1 \subset D_2 \subset D_3 \subset \cdots$, with radii $r_1 < r_2 < r_3 < \cdots$. Then we compute an angular wedge at each node such that the subtree rooted at that node is contain entirely within the angular wedge. The root node is positioned at the origin and the nodes of depth k are arranged on the boundary of the disk D_k. We require the ratio of consecutive radii to be a constant, $\rho = \frac{r_{k+1}}{r_k} > 1$. This guarantees the branches will be spread out nicely. If instead we fixed the difference between consecutive radii, then the ratio $\frac{r_{k+1}}{r_k} \rightarrow 1$ as $k \rightarrow \infty$, and the maximal size of the angular wedges goes to 0. Thus the subtrees of nodes far away from the origin will appear to be arranged along a straight line.

Figure 5 demonstrates the algorithm for computing the angular wedge of a node N, which is on the boundary of the disk D_k. Let β be the angular wedge that has been computed for N. First, we can guarantee no self-intersections by ensuring that all arcs drawn from N to one of its children lie to the right of the tangent to the disk D_k at N. Otherwise, an arc could cross into the interior of the disk D_k, and may intersect an edge of the tree that has already been drawn. To ensure this is not the case we must restrict $\beta \leq 2cos^{-1}(\frac{r_k}{r_{k+1}}) = 2cos^{-1}(\frac{1}{\rho})$. In the figure we show the limiting case where $\beta = 2cos^{-1}(\frac{1}{\rho})$. In our implementation we use $\rho = \sqrt{2}$, thus we restrict $\beta < 2cos^{-1}(\frac{1}{\sqrt{2}}) = \frac{\pi}{2}$. However, one can see that it is only necessary to enforce this condition for the nodes on the boundary of the disk D_1, since we have chosen $\frac{r_{k+1}}{r_k}$ to be constant.

In figure 5 the children of the node N are the nodes N_1, N_2, and N_3. To compute the angular wedges β_i we partition the angle β proportionally to the sizes of the subtrees rooted at each node. If we let n_i be the number of leaves of the subtree rooted at N_i and n the number of leaves of the subtree rooted at N, then we have the following relations:

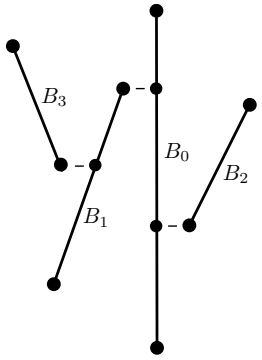


Figure 4. Contour Tree decomposed into branches. The root branch B_0 connects two extrema. B_1 pairs a minimum with a saddle. B_2 and B_3 can be canceled independently.

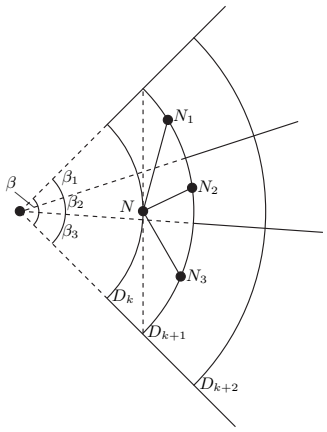


Figure 5. Shows the arrangement used to compute the angular wedges β_1, β_2, and β_3 for the nodes N_1, N_2, and N_3 that are children of N.

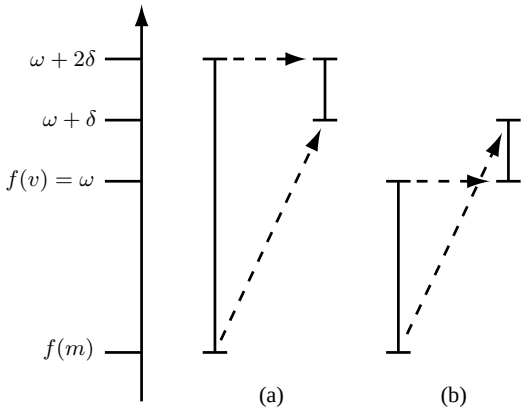


Figure 6. A pictorial representation of the scaling factors used to construct f'. (a) show how the range of the vertices in the region $\mathcal{D}_\delta - \mathcal{D}_\epsilon$ are scaled. (b) shows how the range of the vertices in the region S_ϵ are scaled.

$$n = n_1 + n_2 + n_3$$
$$\beta = \beta_1 + \beta_2 + \beta_3$$
$$n_1 : n_2 : n_3 = \beta_1 : \beta_2 : \beta_3$$

Therefore, we have that $\beta_i = \frac{n_i}{n}\beta \leq \beta$. Since $\beta < \frac{\pi}{2}$ then $\beta_i < \frac{\pi}{2}$ and we can guarantee that the subtree rooted at N_i is free of self-intersections.

To compute the embedding of a hierarchical tree we use the parent-child relationship between branches to construct a rooted tree whose nodes are the branches of the hierarchical tree. Applying the layout algorithm above to this tree produces a planar embedding, which we use for the (x, y)-coordinates of nodes in the hierarchical tree. For each branch we assign these (x, y)-coordinates to all its interior nodes and its unpaired endpoint(s). As stated above the z-coordinate of each node is assigned the function value of the corresponding critical point in the scalar field. The branches are then visualized as "L" shapes, where the base of the "L" connects the branch to its parent along a horizontal line at the height of the paired endpoint.

3 Hierarchical Morse Functions

In this section we develop the tools used to compute a hierarchical representation of the Contour Tree for a given Morse function. While we do not simplify the input Morse function we establish criteria to determine in which cases the simplified Contour Tree corresponds to a Morse function that can be constructed from the input data by cancellations of critical points.

In summary our algorithm is *robust* in the sense that for any input field it constructs a valid branch decomposition of the Contour Tree. In fact, it produces a valid branch decomposition even if the input data has a Reeb graph with loops.

Unfortunately, the simplification is guaranteed, in general, to produce simplifications that have a topological equivalent only if all the saddles in the data merge or split contours. In 3D, for example, there may be pairs of critical points that only change the genus of the contours and that may need to be canceled in pairs. To resolve this, we plan to extend our representation to allow nodes of degree two as in [12]

For simplicity of presentation we assume in the following that the domain $\mathcal{D}$ is a simply connected, compact surface.

Simplification. We develop a multi-resolution framework for distinguishing fine resolution topological features from persistent, coarse resolution structures. There are two operations that can be performed on a Morse function, f, that are known to construct a new Morse function, f': cancellations and handle slides. A cancellation transforms a Morse function into a topologically "simpler" function. Handle slides are more subtle transformations the details of which are not relevant here.

Let $m \in \mathcal{D}$ be an extremum and $v \in \mathcal{D}$ be a saddle point of the Morse function f, such that there is a gradient curve connecting them. We consider the problem of defining a new Morse function f' such that m and v are regular points of f' and all other critical points of f are critical points of f'. When such an f' can be found we say that the pair of critical points (m, v) can be canceled.

A method for computing a sequence of paired critical points, called persistence pairing, was described in [7]. It is based on the definition of the persistent homology groups. The persistence of a pair, (m, v), is defined to be $|f(v) - f(m)|$. One thinks of the lower valued critical point as creating a topological feature and the greater valued one as destroying it. A hierarchy is constructed on these features by sorting them according to their persistence. This hierarchy defines an ideal sequence of simplifications. However, it is known that the critical point pairs cannot, in general, be canceled in this order. The authors introduce the notion of topological obstructions to explain why a cancellation in the sequence cannot be performed.

The algorithm we present constructs a similar hierarchy, but one that defines an order of pairs such that the next pair can always be canceled. Conceptually, we produce a sequence that guarantees for any given pair of critical points

all obstructions are canceled before canceling that pair. In this section, we prove that it is possible to construct such a sequence for any Morse function over $\mathcal{D}$, where $\mathcal{D}$ is a simply connected, closed surface.

Consider a saddle point $v \in \mathcal{D}$ with $f(v) = \omega$. Let C be the contour of the level set $L_f(\omega)$ that contains v. C is the union of two simple closed curves, called *petals*, which intersect at v and do not intersect at any other point in $\mathcal{D}$. A petal of v partitions $\mathcal{D}$ into disjoint regions. The region that contains no other petals of v is said to be enclosed by the petal.

Lemma 1 *Let $f : \mathcal{D} \to \mathbb{R}$ be a Morse function, if f has more than two critical points then it must have at least one saddle point.*

Proof: Since $\mathcal{D}$ is compact, f must have one global maximum and one global minimum. If there is another critical point, it is either a saddle (which proves the theorem) or another extremum. In the latter case the Contour Tree of f has at least three leaf nodes. Since the Contour Tree is connected there must be a node with degree three, which corresponds to a saddle point of f. □

Lemma 2 *Let $f : \mathcal{D} \to \mathbb{R}$ be a Morse function, if f has more than two critical points then there exists a saddle point, $v \in \mathcal{D}$ with a petal that encloses exactly one critical point of f.*

Proof: By lemma 1, f must have a saddle point v_0. Choose a petal of v_0 and the region enclosed by it, M_0. We assume, without loss of generality, that the descending gradient curves starting from to the boundary of $\mathcal{D}_0$ point toward its interior. Thus, there must be a local minimum, m_0, in the interior of M_0. Let f_0 be the restriction of f to $\mathcal{D}_0$. By a symbolic perturbation of the function values on boundary of $\mathcal{D}_0$ we can make v_0 a maximum of f_0 and make all the other points on the boundary of $\mathcal{D}_0$ regular points. If m_0 is the only critical point in the interior of $\mathcal{D}_0$ the theorem is proved. So assume that there are $n_0 > 1$ critical points of f in the interior of $\mathcal{D}_0$. This implies that f_0 has $n_0 + 1 > 2$ critical points, so there must be a saddle point $v_1 \in \mathcal{D}_0$. But v_0 is the only critical point of f_0 on the boundary of $\mathcal{D}_0$ so v_1 must be in the interior of $\mathcal{D}_0$ and therefore it is a saddle point of the entire function f.

Now apply the above construction to v_1 and recursively create a sequence of saddle points $v_0, v_1, \ldots \in \mathcal{D}$. Thus the corresponding sequence of regions $\mathcal{D}_i$, enclosed by the petals of the v_i, satisfy the inclusion relations $\mathcal{D}_0 \supset \mathcal{D}_1 \supset \cdots$. Finally, this implies that the numbers n_i of critical points of f in the interiors of $\mathcal{D}_i$ form a decreasing sequence, $n_0 > n_1 > \cdots$. Since there are only a finite number of critical points and $n_i > 0$ for all i, there must be some number k such that $n_k = 1$. Therefore $v = v_k$ is the required saddle point. □

Lemma 3 *Let $f : \mathcal{D} \to \mathbb{R}$ be a Morse function, v be a saddle point as in lemma 2, and m be the unique critical point enclosed by a petal of v. Then there exists a function f' that cancels the pair (m, v). Moreover, the size of the region where the sign of the gradient of f' differs from that of f can be made arbitrarily small.*

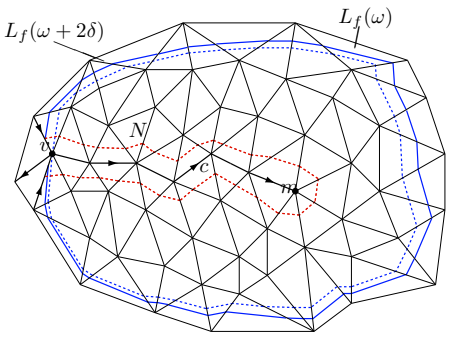


Figure 7. A saddle point v and a minimum m connected by steepest descending path c (arrows). An ϵ-neighborhood of c (red) is the only region where the direction of the gradient is changed. The area inside the curve $L_f(\omega + 2\delta)$ is the only region where the function value is modified.

Proof: First, we distinguish between the topological condition on f' and the geometric one. The topological condition states that f' cancels the pair (m, v). The proof of this fact is a well known theorem of Morse theory and can be found in section 3.4 of [11].

On the other hand the geometric condition states that we can make the region where we must change the sign of gradient flow as small as we like. This condition is slightly stronger than what is typically found in the Morse theory literature. We will demonstrate this is possible by using a triangulation of $\mathcal{D}$. Consider the region of $\mathcal{D}$ shown in Figure 7, such a region exists by lemma 2. Without loss of generality we assume that m is a local minimum. Let c be the steepest descending edge path from v to m. If $f(v) = \omega$ then we subdivide the mesh along the portion of the curve $L_f(\omega + 2\delta)$ shown in Figure 7, for δ small enough. We also subdivide the mesh along the curve N_ϵ, which is defined such that the arcs with endpoints in N_ϵ and c each have length less than ϵ.

The endpoints of the portion of $L_f(\omega + 2\delta)$ drawn in the figure can be connected by following the steepest decent paths that flow into v to form a simple closed curve. Call the region bounded by this curve, $\mathcal{D}_\delta$. Similarly, we can define a simple closed curve by connecting the endpoints of N_ϵ. The region enclosed by this curve will be called $\mathcal{D}_\epsilon \subset \mathcal{D}_\delta$. We now explicitly construct f' by redefining the function values of all the vertices in the region $\mathcal{D}_\delta$ and define $f'(x) = f(x)$ for all $x \notin \mathcal{D}_\delta$. Figure 6 shows how the ranges of the vertices in $\mathcal{D}_\delta - \mathcal{D}_\epsilon$ and $\mathcal{D}_\epsilon$ are scaled. These transformations are reported here:

$$f'(x) = \begin{cases} \dfrac{\delta(f(x) - f(m))}{(\omega + 2\delta) - f(m)} + (\omega + \delta) & x \in \mathcal{D}_\delta - \mathcal{D}_\epsilon \\ \dfrac{\delta(f(x) - \omega)}{f(m) - \omega} + \omega & x \in \mathcal{D}_\epsilon \end{cases}$$

The equation for $x \in \mathcal{D}_\delta - \mathcal{D}_\epsilon$ corresponds to Figure 6(a), which scales the range $[f(m), \omega + 2\delta]$ to the range $[\omega + \delta, \omega + 2\delta]$. On the other hand the range of the vertices $x \in \mathcal{D}_\epsilon$, which is $[f(m), \omega]$, is inverted and scaled to $[\omega, \omega + \delta]$, see Figure 6(b). It is easy to

see that using these equations the sign of the gradient is only changed for points in the region $\mathcal{D}_\epsilon$. Since we can make $\mathcal{D}_\epsilon$ as small as we like, the theorem is proved. Furthermore, the construction demonstrates that the only region where the function value has to be modified is $\mathcal{D}_\delta$. □

4 Multi-Resolution Contour Trees

We present an algorithm for computing a representation of the contour tree that allows linear time access to simplified trees, either by uniform or adaptive simplification. Algorithms for computing the contour tree can be found in [4] and [12]. In both cases the algorithms first make two passes through the data to compute a join tree and a split tree. The degree three nodes of the join tree represent the saddle points where contours are merged, and the those of the split tree represent saddle points where contours are divided. These trees are then merged to construct the contour tree. We use the same approach to construct a hierarchical representation, however, we must store all our trees as branch decompositions and modify the algorithm that merges the join and split trees.

In addition to the basic hierarchical data structure discussed in the previous sections we take into consideration the function value of the vertices associated with each node. Thus we can sort the nodes in a branch by increasing function value. We call the first node the starting node of the branch and the last node the ending node. The length of a branch is defined to be the absolute value of the difference in function value of the endpoints. This value is returned by the function Length(B). Leaf nodes can now be classified as either minima or maxima, by checking if the node is a starting node or ending node respectively. Furthermore, we can now characterize saddle points as either join saddles or split saddles. An interior node is a join saddle if it is the ending point of some branch, whereas it is a split saddle if it is the starting point of some branch. In this characterization a join saddle corresponds to a saddle point of f where two contours merge, and a split saddle to a saddle where one contour divides.

This data structure allows us to make certain queries that we can use to determine if a branch can be simplified. Our algorithm checks the criteria for simplification in a procedure called CanSimplify(G, B), which returns true if the branch B in the graph G represents a valid cancellation. The first criterion for this to be true is that the branch must have no children. If a branch has any children then we say that the child branch is obstructing the parent branch. This condition is necessary but not sufficient for determining if a branch is able to be simplified.

Given a pair of critical points that can be canceled we always think of the first point as creating a topological feature that the second as one destroying it. For example, a minimum creates a new contour. Thus a minimum must be paired with a saddle that destroys that contour, which occurs at a join saddle. Similarly we can see that a maximum must be paired with a split saddle. So the other criterion that must be checked by CanSimplify(G, B) is that the endpoints of B are either a minimum and a join saddle or a split saddle and a maximum.

Once a tree is constructed we can perform several queries on it. First, we include the function GetTree(B) that returns the tree that contains the branch B. For an arbitrary branch decomposition it is possible to have degree 2 nodes. We can check if a node, N, has degree two with the function IsRegular(T, N). If a node is a starting point we can perform the query UpBranch(T, N), which returns the branch that starts at the node. Likewise, we can call DownBranch(T, N) on ending points to access the branch that ends at the node. If CanSimplify(B) returns true for a branch B, then exactly one of it endpoints represents a saddle point. In this case we can access the unique saddle point of the branch by calling GetSaddle(B). Finally, a branch is defined to be a leaf branch if it has no interior nodes and one of its endpoints is a leaf node. The function IsLeafBranch(T, E) returns true if E is a leaf branch.

4.1 Join and Split Trees.

Any of the standard algorithms for computing the join and split trees can be implemented, but the resulting trees must be stored as trivial branch decompositions. In these algorithms every node in each tree represents a critical point. Thus there will be some degree two nodes in each tree, which correspond to saddle points from the other tree.

For completeness we briefly describe the algorithm for constructing the join and split trees that given in [4]. However, this algorithm has been improved upon in [12]. First, the vertices of $\mathcal{D}$ are sorted by function value. The idea is then to keep track of a Union-Find data structure as one sweeps through the vertices in increasing and decreasing order. During the increasing sweep we build the join tree and during the decreasing sweep we build the split tree. We present an algorithm for computing the join tree and describe the differences in the split tree algorithm.

In the pseudo-code given below we make use of a simple Union-Find data structure that includes the functions NewUF(), NewSet(UF, i), Find(UF, i), and Union(UF, i, j). These functions respectively create the data structure, add a new class, return the class containing a given index, and merge two classes. Finally, we require the boolean functions IsMin(v) and IsCritical(v) that return true if v is a local minimum of f and a critical point of f respectively (v is the only argument since $f(v)$ is implicitly determined by its order in a sorted array.)

Algorithm 1. JoinTree
Input: Sorted array of n vertices ($\{v_i\}$) and a triangulated surface ($\mathcal{D}$).
Output: Join tree (JT).

```
1. JT = NewGraph()
2. UF = NewUF()
3. for i = 0 to n − 1 do:
4.     if IsCritical(v_i) then AddNode(JT, i)
5.     if IsMin(v_i) then NewSet(UF, i)
```

6. $i' = \text{Find}(UF, i)$
7. ***for each edge*** $v_i v_j$ ***with*** $j < i$ ***do:***
8. $j' = \text{Find}(UF, j)$
9. ***if*** $j' \neq i'$ ***then*** $\text{AddBranch}(JT, j', i')$
10. $\text{Union}(UF, i', j')$
11. ***return*** JT

The algorithm for constructing the split tree, ST, is almost identical, the only differences are: on line 3 the **for** statement goes from $n-1$ to 0, on line 8 the test $\text{IsMin}(v_i)$ is replace by the test $\text{IsMax}(v_i)$, and on line 9 the edges with $j > i$ are considered. When the ST is constructed in this manner the start of each branch has greater function value than the end. So it is also necessary to reverse the direction of all the branches in the ST in order to make all the saddle points in ST split saddles. This can be done in a subroutine and either included in the algorithm or as a post-processing step before constructing the contour tree.

4.2 Computing the Multi-Resolution CT.

In previous contour tree algorithms the Contour Tree, CT, is constructed form the JT and ST by "peeling off" leaves of the JT and ST and adding them to the CT. This approach uses a queue to store the leaves during processing, which can be removed in any order. Our algorithm uses the same approach, however, we must impose a strong condition on the order in which the leaves are "peeled off." We enforce this condition by using a priority queue such that we always remove the next shortest leaf branch that represents a valid simplification. Once a branch is removed from the queue the adjacent branches in the JT and ST are merged, which is why JT and ST must be stored as branch decompositions. These merges can change the length of branches that are already in the queue. Thus one of the major difficulties in the algorithm is maintaining a valid priority queue. We do this on the fly by simply checking if the top branch of the queue is valid. The condition that must be checked is complex enough to warrant a subroutine, so we present the routine $\text{PopValid}(PQ)$.

There are two possibilities for why the branch at the top of the priority queue is not valid. First of all, the current length of the branch might not be the same as its length when it was entered into the queue. It is possible that the branch was merged with another one, so it could be longer, thus it might have a lower priority than some other branch in the queue. In this case we simply return the branch to the queue with its new priority. Additionally, it might be the case that the for the top branch, B, $\text{CanSimplify}(B)$ returns false. This can come about by removal and merging of branches as well. In this case the branch has become invalidated in a more essential way and we simply remove it from the queue altogether.

The priority queue is a standard data-structure that uses the operations: $\text{Pop}(PQ)$ and $\text{Push}(PQ, B)$, that retrieve the top element of the queue, and push a branch onto the queue respectively. It also supports the test $\text{IsEmpty}(PQ)$ that returns true if there are no elements in the queue. In our case the priority is the length of a branch, B, and $\text{Pop}(B)$ is guaranteed to return the branch with the lowest priority. The priority of a branch, B, when it was entered into the queue can be queried using the function $\text{Priority}(B)$.

Algorithm 2. PopValid
Input: Priority Queue (PQ) of branches
Output: Branch (B) that is a valid simplification

1. $B = \text{Pop}(PQ)$
2. $isValid = false$
3. ***while not*** $isValid$ ***do:***
4. ***if not*** $\text{CanSimplify}(B)$ ***then:***
5. $B = \text{Pop}(PQ)$
6. ***else:***
7. ***if*** $\text{Length}(B) \neq \text{Priority}(B)$ ***then:***
8. $\text{Push}(PQ, B)$
9. $B = \text{Pop}(PQ)$
10. ***else:***
11. $isValid = true$
12. ***return*** B

The procedure $\text{PopValid}(PQ)$ ensures that we can pull the first branch that represents a valid cancellation from the queue. In this way we can ensure that each branch represents a topological simplification of f. It was proved in the previous section that we can always find a simplification, thus $\text{PopValid}(PQ)$ will always return a value as long as the priority is not empty.

For readability we introduce another subroutine of the contour tree algorithm that does the work of "peeling off" a leaf branch. In this routine we make use of the function MergeBranches(B_1,B_2) that merges the branches B_1 and B_2 into the single branch B_1.

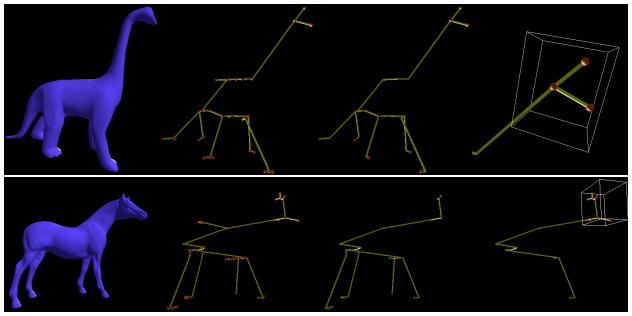
Algorithm 3. PeelOffBranch
Input: Branch (B), Join Tree (JT) and Split Tree (ST)
Output: A branch representing a valid simplification or $null$.

1. $XT = \text{WhichTree}(B)$
2. $N = \text{GetSaddle}(B)$
3. $\text{RemoveBranch}(XT, B)$
4. ***if*** $\text{IsRegular}(JT, N)$ ***then:***
5. $B_1 = \text{DownBranch}(JT, N)$
6. $B_2 = \text{UpBranch}(JT, N)$
7. $\text{MergeBranches}(B_1, B_2)$
8. ***if*** $XT == ST$ ***and*** $\text{CanSimplify}(B_1)$ ***then:***
9. ***return*** B_1
10. ***if*** $\text{IsRegular}(ST, N)$ ***then:***
11. $B_1 = \text{UpBranch}(ST, N)$
12. $B_2 = \text{DownBranch}(ST, N)$
13. $\text{MergeBranches}(B_1, B_2)$
14. ***if*** $XT == JT$ ***and*** $\text{CanSimplify}(B_1)$ ***then:***
15. ***return*** B_1
16. ***return*** $null$

Using the subroutines $\text{PopValid}(PQ)$ and $\text{PeelOffBranch}(B, JT, ST)$ the code for our main algorithm, $\text{BuildContourTree}(JT, ST)$, is relatively simple.

Algorithm 4. BuildContourTree
Input: Join Tree (JT) and Split Tree (ST)
Output: Contour Tree (CT)

1. $CT = \text{NewGraph}$
2. $PQ = \text{NewPQ}$
3. ***for each*** $B \in JT$ ***do:***
4. ***if*** $\text{IsLeafBranch}(B)$ ***and*** $\text{CanSimplify}(B)$ ***then:***
5. $\text{Push}(PQ, B)$
6. ***for each*** $B \in ST$ ***do:***
7. ***if*** $\text{IsLeafBranch}(B)$ ***and*** $\text{CanSimplify}(B)$ ***then:***



(a) (b) (c) (d)

Figure 8. (a) Two geometric models of a dinosaur (top row) and a horse (bottom row). (b) The contour trees of each model, where the function is the displacement along the vertical axis. (c) Simplified trees with persistence value 0.02. (d) Adaptively simplified trees showing detail around the head.

8. Push(PQ, B)
9. ***while not*** IsEmpty(PQ) ***do:***
10. $B_{top} =$ PopValid(PQ)
11. AddBranch(CT, B_{top})
12. $B_{next} =$ PeelOffBranch(B_{top}, JT, ST)
13. ***if not*** $B_{next} \neq null$ ***then*** Push(PQ, B_{next})
14. ***return*** CT

It is clear from the discussion in this and the previous section that this algorithm produces a multi-resolution contour tree, such that each branch represents a valid topological simplification. We can now define an order on the branches that allows one to extract a contour tree after any number of simplifications in linear time. First, we define the persistence of a branch to be the greater of its length and the persistence of each of it children. This definition differs from the definition of persistence given in [7] because it takes into consideration the topological obstructions. Thus a pair of critical points is never assigned a persistence value that is less than any of its obstructions.

The same analysis as in [3] can be used to show that the complexity of BuildContourTree is $O(n \log n)$ where n is the number of nodes in JT and ST. Using a simple FIFO queue instead of a priority queue would yield a complexity of $O(n)$ but with the risk of building an unbalanced tree. In practice this does not seem to be a problem and a linear queue may be advisable for a faster implementation with lighter data-structures.

The branch decomposition representation of the CT allows for uniform or adaptive refinement of the tree. Uniform simplification is achieved by interrupting the drawing process when the first branch with persistence less than a specified value is reached. Adaptive simplification is almost as easy, before each branch is visualized it is tested to see if it satisfies the adaptive criterion (see Figure 8).

References

[1] Peer-Timo Bremer, Herbert Edelsbrunner, Bernd Hamann, and Valerio Pascucci. A multi-resolution data structure for two-dimensional Morse functions. In *Proceeding of IEEE Conference on Visualization*, pages 139–146, October 2003.

[2] H. Carr and J. Snoeyink. Path seeds and flexible isosurfaces - using topology for exploratory visualization. In *Proceeding of IEEE TCVG Symposium on Visualization (VisSym '03)*, pages 49–58, Grenoble, Fr, May 2003.

[3] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 918–926, January 2000.

[4] Hamish Carr, Jack Snoeyink, and Ulrike Axen. Computing contour trees in all dimensions. *Computational Geometry*, 24(2):75–94, February 2003.

[5] Kree Cole-McLaughlin, Herbert Edelsbrunner, John Harer, Vijay Natarajan, and Valerio Pascucci. Loops in reeb graphs of 2-manifolds. In *ACM Symposium on Computational Geometry*, pages 344–350, July 2003.

[6] Giuseppe di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

[7] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Proceeding of The 41st Annual Symposium on Foundations of Computer Science*. IEEE, November 2000.

[8] A. T. Fomenko and T. L. Kunii, editors. *Topological Modeling for Visualization*. Springer-Verlag, Tokyo, 1997.

[9] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii. Topology matching for full automatic similarity estimation of 3d shapes. In *ACM SIGGRAPH*, pages 203–212, 2001.

[10] E. Kleiberg, H. van de Wetering, and J.J. van Wijk. Botanical visualization of huge hierarchies. In *Proceedings IEEE Symposium on Information Visualization (InfoVis'2001)*, pages 87–94, 2001.

[11] Y. Matsumoto. *An Introduction to Morse Theory*. AMS, 1997.

[12] Valerio Pascucci and Kree Cole-McLaughlin. Efficient computation of the topology of the level sets. In *IEEE Visualization*, pages 187–194, October 2002.

[13] G. Reeb. Sur les points singuliers d'une forme de pfaff completement integrable ou d'une fonction numerique. *Comptes Rendus Acad. Sciences Paris*, 222:847–849, 1946.

[14] Y. Shinagawa, T. L. Kunii, H. Sato, and M. Ibusuki. Modeling the contact of two complex objects: With an application to characterizing dental articulations. *Computers and Graphics*, 19:21–28, 1995.

[15] Y. Shinagawa and T.L. Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 11:44–51, November 1991.

[16] S. Takahashi, G. M. Nielson, Y. Takeshima, and I. Fujishiro. Topological volume skeletonization using adaptive tetrahedralization. In *Proceeding of Geometric Modeling and Processing*, 2004. to appear.

[17] S. Takahashi, Y. Takeshima, and I. Fujishiro. Topological volume skeletonization and its application to transfer function design. *Graphical Models*, 66(1):24–49, 204.

[18] Alexandru Telea and Jarke J. van Wijk. Simplified representation of vector fields. In *Proceedings of the conference on Visualization '99*, pages 35–42.

[19] Xavier Tricoche, Gerik Scheuermann, and Hans Hagen. A topology simplification method for 2d vector fields. In *Proceedings of the conference on Visualization '00*, pages 359–366. IEEE Computer Society Press, 2000.

[20] Edward R. Tufte. *Envisioning Information*. Graphics Press LLC, Cheshire, Connecticut, 1990.

[21] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the Symposium on Computational Geometry (SCG-97)*, pages 212–220, June 1997.

[22] Gunther H. Weber and Gerik Scheuermann. *Automating Transfer Function Design Based on Topology Analysis*. 2004.